COBB

INSIDE DOS

Tips & techniques for MS-DOS & PC-DOS

February 1993
Vol. 4 No. 2

Guarding against the hidden DOS trap

By Van Wolverton

Deleting a file by mistake is definitely not good for your self-esteem. The act can create feelings of remorse, self-loathing, even despair (well, OK, it would have to be a pretty important file to inspire despair, but you get the idea). We've all done it; none of us is immune to the occasional fit of pilot error. And because we don't seem able to grow into perfection, programmers have come up with tools we can use to recover from our errors.

At first, there were the Norton Utilities and their magic Undelete program, which let us recover a deleted file. Similar programs followed Norton's offering—PC Tools and the Mace Utilities chief among them. Now, starting with version 5, DOS itself offers an UNDELETE command, which lets us undelete a file if we act before storing any other files on the disk.

With the addition of UNDELETE, UNFORMAT, and the improved FORMAT command, DOS has, in fact, become much safer to use over the last few versions. But a serious trap remains for the unwary, a way of losing a valuable file without hope of recovery and not even knowing that it's gone until you try to use it.

The culprit is the COPY command. (By the way, XCOPY also harbors this trap.) If you tell DOS to copy a file over another file—to make a copy of a file and give the copy the same name as an existing file—DOS dutifully replaces the existing file with the copy of the different file without giving you a chance to confirm your intention. That's fine when you're replacing your to-do list with tomorrow's tasks or storing a revised version of a proposal you've been working on for a couple of weeks. But if you inadvertently copy a two-line batch file over a 10-page spreadsheet, the spreadsheet is gone forever. No undelete will recover it for you because, as far as DOS is concerned, the file wasn't deleted—its contents were changed.

The Shell can help

So how do you protect yourself against losing a file this way? The first line of defense, of course, is: Don't press [Enter] after you type any command that could conceivably damage or destroy a file. If you've been maintaining this vigilance whenever you type a DELETE command, do it whenever you type a COPY command, too, especially when you use wildcards in the source or destination filenames.

But we're fallible, and it's tough to be eternally vigilant. Another, surer way to guard against this insidious way of losing files is to use the DOS Shell to manage your files. First, select Confirmation... from the Options menu, then make sure that the Confirm on Replace option is selected. If it's not already selected, you can tab to the check box and press the [Spacebar] to place an X inside it, then click OK or press [Enter] to save the setting. Selecting the Confirm on Replace option tells the Shell to warn you whenever you try to copy a file over an existing one. Not only does the Shell ask you to confirm such a replacement, but the message asking for confirmation shows you the date and size of both files involved, increasing the likelihood that you'll spot an error before telling the Shell to replace the file.

"Fixing" DOS with batch files

But what if you don't have version 5 or don't want to use the Shell? Like most other DOS problems, this one can be solved—well, perhaps "alleviated" is a bit more accurate—by a batch file.

The SAFECOPY.BAT batch file, shown in Figure A on the next page, gives you a limited form of the confirm-on-replace protection offered by the Shell. The batch file

IN THIS ISSUE

uses two IF statements to check whether the destination file exists before actually copying the source file to the destination. If the destination file exists, the batch file displays a warning message instead of copying the file. On the other hand, if the destination file doesn't exist, the batch file copies the file.

Figure A

```
@echo off
rem SAFECOPY.BAT helps prevent DOS from overwriting
rem files when you copy them.
echo.
if not "%2"=="" goto :OK
echo•You must enter a destination.
goto :END
:OK
if exist %2 goto :NOTSAFE1
if exist %2\%1 goto :NOTSAFE2
copy %1 %2
goto :END
:NOTSAFE1
echo The file %2 exists and would be replaced by %1.
echo•FILE NOT COPIED.
goto :END
:NOTSAFE2
echo A file named %1 already exists in the %2 directory.
echo•FILE NOT COPIED.
:END
echo.
```

SAFECOPY.BAT helps prevent the COPY command from overwriting files.

How SAFECOPY.BAT works

SAFECOPY.BAT, like most batch files, begins with a command to turn echoing off and REM commands to explain what the file does. The *echo.* line simply echoes a blank line to the screen. Then, an IF statement provides the first "test" in the batch file. The statement

```
if not "%2"=="" goto :OK
```

jumps to the :OK label if you typed a second parameter (%2) when you ran the batch file. If you didn't type a second parameter, the batch file executes the next two statements:

```
echo•You must enter a destination.
goto :END
```

The special symbol represented by the bullet is the ASCII 7 character (also known as [Ctrl]G or ^G). When echoed, this character will cause your computer to beep. Using the DOS Editor, you can create the special character by pressing [Ctrl]P, then [Ctrl]G. The Editor will display the small diamond or bullet character. In SAFECOPY.BAT, the special character causes the batch file to beep when it echoes the message *You must enter a destination* and then quit.

The :OK section that follows is the heart of the batch file. The section begins with the following IF statement:

```
if exist %2 goto :NOTSAFE1
```


INSIDE DOS™

Inside DOS (ISSN 1049-5320) is published monthly by The Cobb Group.

Prices
Domestic ..... $49/yr. ($6.00 each)
Outside U.S. ..... $69/yr. ($8.50 each)

Phone
Toll free ..... (800) 223-8720
Local ..... (502) 491-1900
Customer Relations Fax ..... (502) 491-8050
Editorial Department Fax ..... (502) 491-4200

Address
You may address tips, special requests, and other correspondence to:

The Editor, *Inside DOS*
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

For subscriptions, fulfillment questions, and requests for bulk orders, address your letters to:

Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

Postmaster
Second class postage is paid in Louisville, KY. Send address changes to:

Inside DOS
P.O. Box 35160
Louisville, KY 40232

Back Issues
To order back issues, call Customer Relations at (800) 223-8720. Back issues cost $6.00 each, $8.50 outside the U.S., and you can pay with MasterCard, VISA, Discover, or American Express, or we can bill you. Please identify the issue you want by the month and year it was published. Customer Relations can also provide you with an issue-by-issue listing of all the articles that have appeared in *Inside DOS*.

This statement checks to see if the file represented by the second parameter already exists. This test is useful only when you type both the path and name of the destination file as the second parameter. For example, this test would prevent you from overwriting a TOKENS.DOC file in the C:\SUBWAY directory if you entered the following command:

```
C:\>safecopy \docs\tokens.doc \subway\tokens.doc
```

(We've omitted C: from the path since all the files are on the C: drive.) If SAFECOPY.BAT found that the C:\SUBWAY\TOKENS.DOC already existed, it would skip to the :NOTSAFE1 section, display the message *The file \subway\tokens.doc exists and would be replaced by docs\tokens.doc.*, beep, and then display the message *FILE NOT COPIED*. Usually, this test is useful when you rename a file as you copy it. For example, suppose you want to rename the TOKENS.DOC file to SALES93.DOC when you copy it to the C:\SUBWAY directory. You can safely do this by entering the command

```
C:\>safecopy \docs\tokens.doc \subway\sales93.doc
```

Again, if a file named C:\SUBWAY\SALES93.DOC (that is, %2) exists, SAFECOPY.BAT will present an error message and quit.

However, if no file named %2 exists, SAFECOPY.BAT will proceed to the next IF statement:

```
if exist %2\%1 goto :NOTSAFE2
```

This statement safeguards you when you copy a file from the current directory to another directory, without specifying a new name for the file. Returning to our example, suppose you are in the C:\DOCS directory and you copy the file named 930120.DOC to the C:\SUBWAY directory with the following command:

```
C:\DOCS>safecopy 930120.doc \subway
```

If a file named 930120.DOC exists in the C:\SUBWAY directory, SAFECOPY.BAT will jump to the :NOTSAFE2 label, display the message *A file named 930120.doc already exists in the \subway directory*, beep, and display the message *FILE NOT COPIED*.

On the other hand, if no such file exists in the destination directory, SAFECOPY.BAT will execute the next two commands:

```
copy %1 %2
goto :END
```

The first command copies the file you specified, and the second command quits the batch file.

Replacing the COPY command using DOSKEY

One problem with SAFECOPY.BAT is remembering to use it instead of the COPY command. Another problem is that, by itself, SAFECOPY.BAT won't work with wildcards. You can avoid both of these potential problems by creating a COPY macro. Although COPY is an internal DOS command, you can replace it by defining a DOSKEY macro with the same name. You can replace COPY with SAFECOPY.BAT by entering the following DOSKEY command:

```
C:\>doskey copy=for %a in ($1) do call safecopy %a $2
```

By phrasing the macro as a FOR statement, you'll be able to use wildcards when you copy files from the current directory to another directory. Let's take a closer look at the statement.

First, you may recognize $1 and $2 as the DOSKEY symbols for parameters. These symbols will represent the %1 and %2 parameters you type after *copy*. But we use the symbol %a to represent the *currently* selected member of the set represented by $1. For example, if you enter:

```
C:\SUBWAY>copy *.txt c:\docs
```

the DOSKEY COPY macro will define all TXT files as its set. In this case, the command will become

```
for %a in (*.txt) do call safecopy %a c:\docs
```

Each file with the TXT extension will, in turn, replace the %a variable, thereby carrying out the DO CALL SAFECOPY command for one file at a time. For example, the DOSKEY COPY macro might process the file GRAFFITI.TXT as

```
call safecopy graffiti.txt c:\docs
```

The CALL command ensures that DOS will return control to the COPY macro after running the SAFECOPY command for each file.

If you copy a number of files, you'll see the CALL commands scroll by as DOS runs SAFECOPY.BAT for each file that meets the specification. Normally, you'll see the message *1 file(s) copied* after each command. When SAFECOPY finds a filename that already exists, however, it will display the message *0 file(s) copied*.

To replace the COPY command permanently, put the DOSKEY command in your AUTOEXEC.BAT or MACROS.BAT file. (We described how to create a MACROS.BAT file in the article "Storing Your Macros and Automatically Loading Them at Bootup," which appeared in the September 1991 issue of *Inside DOS*.) Remember, if you put the DOSKEY command in a batch file, you'll have to double the % signs before the letter *a*.

The form of the command you can use in a batch file, therefore, is:

```
doskey copy=for %%a in ($1) do call safecopy %%a $2
```

If you'd like to return to the standard version of the COPY command, just enter this DOSKEY command:

```
doskey copy=
```

Nobody's perfect

Although substituting SAFECOPY.BAT for the regular COPY command will protect you from overwriting files in most instances, the technique is not perfect. The IF statements SAFECOPY.BAT (and, consequently, the COPY macro) uses to check for files aren't foolproof. In fact, the IF tests work reliably only under two conditions:

- when you specify a complete file spec (path and name) as the destination, as in

  ```
  C:\>copy \utils\dial.bat \batch\dial.bat
  ```

- when you copy a file (or files) from the current directory and specify only the name of the destination directory, as in

  ```
  C:\UTILS>copy *.bat \batch
  ```

Although these conditions will probably cover most of your file copying needs, SAFECOPY.BAT and the COPY macro can't prevent you from overwriting a file when you specify the complete path and name of the file you want to copy, but only the destination directory. For example, when you enter the command

```
C:\>copy \utils\dial.bat \batch
```

the COPY macro will overwrite a C:\BATCH\DIAL.BAT file, if one exists. This occurs because SAFECOPY.BAT looks only at the parameters %1 and %2—it can't separate path names from filenames. When it tests for the existence of the file %1 in the C:\BATCH directory, it looks for a file named %2\%1, which is, in this case, \BATCH\\UTILS\DIAL.BAT. This file specification is not what you're looking for—C:\BATCH\DIAL.BAT—but SAFECOPY.BAT can't make that logical distinction.

So, if you're using a COPY macro assigned to SAFECOPY.BAT, just remember to change to the directory containing the files you want to copy. Or, if you're copying only one file at a time, you can instead specify the complete path and name as the file's destination. But if you want foolproof protection for the COPY command, use the DOS Shell with the Confirm on Replace option turned on.

Contributing editor Van Wolverton is the author of the best-selling books Running MS-DOS 5 *and* Supercharging MS-DOS. *Van, who has worked for IBM and Intel, currently lives in Alberton, Montana.*

DEBUG TECHNIQUE

Creating commands for controlling NUM LOCK and CAPS LOCK

Contributing editor David Reid submitted the DEBUG scripts used in this article.

Several *Inside DOS* subscribers have asked if we have a technique to turn NUM LOCK on or off when they boot their PCs. Others have asked if we have a technique for turning CAPS LOCK on or off as part of a batch file, without having to press the [CAPS LOCK] key. Although these capabilities are not built into DOS 5, it's fairly easy to create commands for controlling CAPS LOCK and NUM LOCK on IBM-compatible keyboards. As we'll show you in this article, you do this by first creating a script file, then sending the script file through DOS' DEBUG utility to create the command (COM) files. (If you're unfamiliar with DEBUG, you might want to check out the article "Getting to Know the DEBUG Utility" on page 6.) Let's begin by looking at the script files.

The script files

Both of the script files we'll show you create two commands: one to turn the feature on and one to turn it off. NUMLOCK.SCR, the script shown in Figure A, creates NUMLKON.COM and NUMLKOFF.COM. The script in Figure B, CAPSLOCK.SCR, creates CAPSLKON.COM and CAPSLKOF.COM. You can create either the NUMLOCK.SCR or CAPSLOCK.SCR script by typing each line of the corresponding script file exactly as shown. As with batch files, you must use the DOS Editor or a word processor that allows you to save files in an ASCII, text-only format.

You'll want to keep the new commands in a directory on your path so that DOS can find and run them, no matter which directory is current. We suggest you create both the script file and resulting command files in a directory that's already on your path, such as C:\BATCH or C:\DOS. You can name the script file as you run the DOS Editor by changing to the directory you've chosen and typing *edit*, followed by the name of the script file.

For example, to create the NUMLOCK.SCR script file in your C:\BATCH directory, you can type *cd \batch* to change to the directory. Then, you can run the Editor with the following command:

```
C:\BATCH>edit numlock.scr
```

The DOS Editor will start and create a new file called NUMLOCK.SCR. In the empty file, you begin the script by typing *n numlkon.com*. After you make sure the line is correct, press [Enter] to go to the second line.

You continue entering the script line by line. As you can see, lines 13, 17, and 30 are blank. When you reach them, just press [Enter] to go to the next line. Otherwise, don't leave any blank lines in the script.

Figure A

```
n numlkon.com
a 100
mov ax,0040
mov ds,ax
mov si,0017
cli
mov al,[si]
or al,20
mov [si],al
sti
mov ax,4c00
int 21

rcx
15
w

n numlkoff.com
a 100
mov ax,0040
mov ds,ax
mov si,0017
cli
mov al,[si]
and al,df
mov [si],al
sti
mov ax,4c00
int 21

rcx
15
w
q
```

The NUMLOCK.SCR script lets you create the NUMLKON.COM and NUMLKOFF.COM command files.

As you type the file, note that there are no spaces after commas. For example, the *mov ax,0040* command on line 3 contains a space after *mov*, but no spaces after *ax* or the comma. Also, when you enter NUMLOCK.SCR, be careful to type zeros, not uppercase *O*s.

Figure B

```
n capslkon.com
a 100
mov ax,0040
mov ds,ax
mov si,0017
cli
mov al,[si]
or al,40
mov [si],al
sti
mov ax,4c00
int 21

rcx
15
w

n capslkof.com
a 100
mov ax,0040
mov ds,ax
mov si,0017
cli
mov al,[si]
and al,bf
mov [si],al
sti
mov ax,4c00
int 21

rcx
15
w
q
```

The CAPSLOCK.SCR script contains the instructions for creating two command files: CAPSLKON.COM and CAPSLKOF.COM.

Using DEBUG to create the commands

Once you've finished typing and saving the script file, you're ready to use the DEBUG utility to create the commands. You do this by typing *debug*, a less-than sign (<), followed by the name of the script file. When you press [Enter], DOS will redirect the script file into the DEBUG utility, which will compile the script into two command files.

Returning to our example, suppose you're ready to use the NUMLOCK.SCR script file to create the NUMLKON.COM and NUMLKOFF.COM commands. All you have to do is tell DEBUG to process the instructions you've typed in the script. To do this, you issue the following command:

```
C:\DOS>debug < numlock.scr
```

If you've typed all of the instructions correctly, DEBUG will quickly create the two command files, displaying the message *Writing 15 bytes* as it creates each one. (If DEBUG doesn't stop in a minute or so, you will have to reboot your computer and edit the script NUMLOCK.SCR.) Once DEBUG quits, you can check to see that the new commands work. First, turn on NUM LOCK (if it's not already turned on) by pressing the [NUM LOCK] key, then type

```
C:\DOS>numlkoff
```

and press [Enter]. You should see the NUM LOCK indicator light on your keyboard go off. You can verify that NUM LOCK is turned off by pressing 6 on the numeric keypad. When NUM LOCK is turned off, pressing 6 will move the cursor one character to the right.

Next, test the NUMLKON command by entering

```
C:\DOS>numlkon
```

The NUM LOCK indicator light on your keyboard should turn on. Again, you can verify that NUM LOCK is on by typing numbers with your numeric keypad.

If one or both of these commands doesn't work, you'll need to re-edit NUMLOCK.SCR, looking closely for a typo or missing line. Once you've made the correction, issue the command *debug < numlock.scr* again to re-create the commands.

Although we've used the NUM LOCK commands for our example, you can create and test the CAPSLKON and CAPSLKOF commands in much the same way.

Using the commands

As we've seen in our testing, you can use the commands you create by entering them at the DOS prompt. Obviously, this isn't a very practical way of using the commands, since it's much easier to simply press the [NUM LOCK] and [CAPS LOCK] keys. The best use for these commands is to control NUM LOCK or CAPS LOCK through a batch file. For example, if you want your computer to turn off NUM LOCK when it boots, simply place

```
numlkoff
```

at the end of your AUTOEXEC.BAT file. Then, whenever you boot your computer, NUM LOCK will automatically be turned off, and you can use the numeric keypad to move the cursor.

Or, suppose you want to turn CAPS LOCK on whenever you run a terminal emulation program. You can place the CAPSLKON command at the beginning of a batch file that runs the program (TERMINAL.EXE), then place CAPSLKOF at the end, as shown below:

```
@echo off
capslkon
cd \term
terminal
capslkof
```

Conclusion

Some DOS users prefer a mode for NUM LOCK or CAPS LOCK that's different from their system's default mode. In this article, we've shown you how to create commands to turn NUM LOCK and CAPS LOCK off and on.

DEBUG BASICS

Getting to know the DEBUG utility

In the accompanying article, "Creating Commands for Controlling NUM LOCK and CAPS LOCK," we show you how to create and compile script files using DEBUG. Judging from the questions we've received from *Inside DOS* subscribers, you might be curious about what DEBUG does and how it relates to DOS. In this article, we'll give you some background on the DEBUG utility and review some general guidelines for using it.

Talking to a machine

DEBUG scripts are indecipherable to most human beings. That's because DEBUG's main purpose is to talk to your PC at a very basic level. Instead of using a human language (such as English) or even a higher-level programming language (such as QBasic), you must use the Assembler language in DEBUG. Then, you make DEBUG convert the Assembler language instructions to the most fundamental language of all: machine language.

The goal of all this is usually to create a command file. As you may know, command files always have the COM extension. But you can't just slap a COM extension on any file and expect DOS to carry out its instructions. Because these files address your computer's memory in a special way, they must be compiled into machine language—and that's why you need DEBUG to create them.

Although we'll focus on creating commands with DEBUG, the utility has some other uses. Some programmers use DEBUG to look at a particular segment of memory—a useful tool when you need to debug a program. Others use DEBUG to make minor changes to program files. That's fine if they've created the programs themselves, but tweaking commercial programs with DEBUG violates most software licensing agreements.

The advantages of scripts

Although the DEBUG utility will let you create a command file by typing each line at the DEBUG prompt, we prefer creating a script file instead. The script method is much more forgiving than working directly in DEBUG.

As with batch files, you must use the DOS Editor, Edlin, or a word processor that produces plain, ASCII-only text files when you create a DEBUG script. DEBUG won't forgive even a one-character typo, so be sure to check your script before saving it. In *Inside DOS* articles, we try to point out any characters that may be confusing—such as zeros and Os—in the DEBUG scripts we show you.

Once you've saved the DEBUG script and returned to DOS, you're ready to use DEBUG to compile it into a command. You do this by redirecting the script file into the DEBUG program. The syntax for this takes the form

```
C:\BATCH>debug < filename.scr
```

where *filename.scr* is the name of the script file you created. Be careful to type *debug*, followed by the less-than sign (<) *before* typing the filename. When you press [Enter], DEBUG will transform the instructions you typed in the script file into commands that your PC can recognize. If all goes well, DEBUG will run for a few seconds as the instructions scroll across the screen. Then, the utility should stop and present the message *Writing xx bytes*, where *xx* is the number of bytes in the resulting command file. If DEBUG doesn't quit, there is an error in the script.

Troubleshooting DEBUG scripts

If DEBUG is unable to compile your script, you may be able to see which line is causing the error. Here are a few common causes for errors in compiling commands:

- If errors begin in the first or second line of the file, check to see that you entered the *a 100* command properly.
- If the DEBUG utility just keeps running, check to see that you've included the Q command (Quit) at the end of the script file.
- If DEBUG quits, but it didn't create the command, make sure you typed the W command (Write) near the end of the script.

Unfortunately, DEBUG errors are rarely so easy to spot. Usually, you'll find that you've made a typo in one or two lines.

DEBUG helps you find typos by placing an error message below the line causing the problem. For example, if you type *mev* instead of *mov* in a script file, DEBUG will display an error message similar to the one below:

```
2B54:0109 mev al,[si]
          ^ Error
```

You might find it difficult to spot these error messages as they scroll by. You can capture any error messages DEBUG generates by telling DOS to echo the screen display to the printer. You can turn on screen echoing by pressing [Ctrl][Print Screen]. Then, issue the DEBUG command to compile the script again. After DEBUG finishes, press [Ctrl][Print Screen] again to stop echoing the screen display to the printer. (On laser printers, you'll then have to press the form feed button to eject that page.)

Even when DEBUG successfully compiles a script, the commands you create can sometimes cause errors when you try to use them. Commands compiled with an error can make your PC freeze. Fortunately, you can clear the command from memory by turning off your PC, then turning it on again. (Waiting about 20 seconds before turning your PC on again can help protect its circuits.) Since you know the command has an error in it, delete the new command file. You can leave the script file, however, and examine it for errors. Since the script compiled, it contains valid instructions, but they're probably arranged in the wrong order. Check to see if you've accidentally omitted, duplicated, or transposed a line. Also, check any numbers in the script to make sure they're not transposed.

Conclusion

In this article, we've given you a bit of background information on what DEBUG does and how to use it. We've also looked at some ways to troubleshoot DEBUG scripts when DEBUG fails to compile a script or when the resulting command doesn't work properly.

The following articles show how to create commands with the DEBUG utility:

"Soliciting a Reply," March 1992

"A Better Way to Guard Against Unauthorized Access," May 1991

"Clearing Memory-Resident Programs," May 1991

"Adding a GETDRIVE Command to DOS," March 1991

"A Better Way to Solicit Input in a Batch File," December 1990

See the masthead on page 2 for information on ordering back issues.

MAINTENANCE TIP

Preparing for battery failure will help you get back to work quickly

Many PC users don't realize that their computers use a battery until the machine starts behaving erratically. When this happens, some people are relieved to find out that their computers require only a fresh battery—usually selling for under $10—to get up and running again. But others complain of the time they lose reconfiguring their computers after the battery failure. As we'll show you in this article, you can avoid much of that down time by preparing for the day your PC battery fails. But first, let's look at why your PC needs battery power.

Memory power

Since IBM introduced the PC-AT in 1986, most personal computers have used batteries for two important tasks:

- to keep the computer's internal system clock running when the computer is turned off
- to power the memory that stores the computer's configuration settings

Generally, you can assume that your PC uses a battery if it keeps track of the time and date. If your PC is an older model that doesn't use a battery, you have to reset the time and date every time you turn on your computer.

Of course, your computer's battery (or, in some cases, batteries) will eventually run down and need to be replaced. You'll be able to tell when the battery is running down because the computer will lose track of the time and date.

Your computer will also "forget" how it's configured. This happens because many PCs store important information in a type of memory called CMOS (which is short for *complementary metal-oxide semiconductor*). The information stored in CMOS memory—such as the type of disk drives installed, the amount of base memory and extended memory, and the type of display—configures your PC on a more fundamental level than the CONFIG.SYS file does. Because of this, you can boot your PC without a CONFIG.SYS file, but you can't boot the PC when the CMOS configuration settings are missing.

As you may know, the random access memory (RAM) in your PC can hold information only while the power is turned on. That's why a power failure can cause you to lose changes you've made to a document since last saving it. Like RAM, CMOS memory requires electricity to store information. However, CMOS memory needs only the small amount of current a battery provides to store the configuration information. When the battery fails, the CMOS memory loses the configuration information stored in it. When this happens, you might turn on your PC and receive a message such as *Press [F1] to run Setup* or *Run Setup from Diagnostics diskette*. (We'll discuss how you can restore this information in the next section.)

Dealing with battery failure

Now that we've given you some background on why the battery is important, let's look at some things you can do to deal with its inevitable failure. Depending on how often you power down your machine, most computer batteries last at least three years. Some computers can go for years on the same battery, while others seem to need replacements at least once a year. As a rule of thumb, you should probably replace the battery when you've used it for three years—you may be able to ward off an inconvenient failure.

Your computer's instruction manual should list which battery you need. If you can't find the information in the manual, try calling a local retailer that carries the same brand of computer. A retailer should be able to look up the battery your PC uses and tell you if one is in stock, or order one, if necessary. If no local store carries your brand of PC, call the PC's manufacturer, which should be able to send you a battery and bill you or charge it to your credit card.

If your battery fails, you should be able to deal with it fairly easily by following these steps:

- record your computer's configuration settings
- remove the cover and replace the battery
- turn on your computer and restore the configuration settings

Recording the configuration settings

If you're not prepared, resetting your system's configuration is one of the most frustrating tasks you'll face as a PC user. Few of us know off the top of our heads how many cylinders our hard disk drives have. But that's just the kind of information you'll need to know to reset your configuration.

Fortunately, most systems allow you to record configuration information quickly. But you'll need to record the settings now—before the battery loses power—and store them in a safe place. Once the battery begins to fail, it will be too late to retrieve the settings. Let's look at the two most common ways to record system information.

Pressing keys during boot up

Many PCs—including models from Dell, Zenith, and Zeos—allow you to press a special key combination to access the configuration settings screen. Your PC's manual should tell you exactly which keystroke to use. Most PCs require you to hold down the [Ctrl] and [Alt] keys while you press another key. For example, a Zeos 386 we tested will display the configuration screen if we press [Ctrl][Alt][Esc] during boot up. On the other hand, a Dell 486P/33 we tested requires pressing [Ctrl][Alt][Enter], then [F2]. Some Zenith models use the [Ctrl][Alt][Insert] keystroke, while other PCs use the [F1] key.

Whichever key combination your system requires, the trick is to press the key combination at a certain point as your PC boots up. Usually, that's after the computer displays a memory check. You'll need to press the key combination after this memory check, but before your computer runs your CONFIG.SYS file. We've found that pressing the key combination as the computer checks its diskette drives usually works. If your PC doesn't stop the first time you press the key combination, quickly try again as it continues to boot.

Once you've figured out how to bring up your computer's configuration settings, you'll need to record them. In most cases, you can place your printer online and press [Shift][Print Screen] to send all of the information to the printer. While you may not have an attractive border, printing the screen lets you quickly record the information you need. For example, Figure A shows the configuration settings for our Zeos 386. *(Please note that your configuration settings will almost certainly be different from ours.)*

Figure A

```
                    Phoenix Technologies Ltd.   Version
                    System Configuration Setup  4.03 02

Time:   15:44:01
Date:   Mon Nov 23, 1992

Diskette A:           5.25 Inch, 1.2 MB
Diskette B:           3.5 Inch, 1.44 MB     Cyl  Hd   Pre   LZ   Sec Size
Hard Disk 1:          Type 41               917  15    -1   918  17  114
Hard Disk 2:          Not Installed
Base Memory:          640 KB
Extended Memory:      3456 KB
Display:              VGA/EGA
Keyboard:             Installed
CPU Speed:            Fast
BIOS Shadow:          Disabled
Video Shadow:         Disabled

Coprocessor:          Not Installed

Up/Down Arrow to select. Left/Right Arrow to change.
F1 for help. F10 to Exit. Esc to reboot.
```

You can press [Shift][Print Screen] to record your configuration settings, as we've done for our Zeos 386.

If you can't print the screen, you'll have to write down the information. You'll see all kinds of settings for the date and time, diskette drives, hard disks, base memory, extended memory, display type, and so forth. Make absolutely certain you record all the detailed information regarding your hard drive—if your computer permanently loses this information, you may not be able to access that drive. Even worse, if you later key in the wrong information for your hard drive, you can permanently damage that drive.

Be careful not to change any of the information on the configuration screen. Once you've recorded all of the settings, press the key combination the screen tells you to use for exiting. In some cases, your PC may display a message like *Time of day clock stopped* when you reboot. You can respond by pressing the indicated key (usually [F1]) to continue rebooting.

Running Setup from a diskette

Other types of PCs store the configuration program on a setup diskette. Again, your PC's manual will give you the details about which diskette contains the program and how to run it. But let's look at one method of running a configuration program from a diskette.

For example, if you're using an IBM PC-AT, you'll specify configuration settings by inserting the Setup and Diagnostics disk into drive A and rebooting your machine. (The Setup and Diagnostics diskette is stored in a plastic sleeve near the end of your computer's setup documentation.) When the startup screen appears, choose Run Setup to bring up the computer's configuration settings.

Once you have the settings onscreen, you can try pressing [Shift][Print Screen] to print the information. If that doesn't work, you'll have to record the settings.

Removing the cover and replacing the batteries

Replacing the battery in your computer isn't as easy as replacing the batteries in a flashlight, but it's still pretty simple. Your computer owner's manual should offer step-by-step instructions for replacing the computer's battery. Essentially, all you have to do is

- turn off and unplug your computer
- remove the screws from the computer's cover
- remove the old battery (or batteries)
- install the new battery (or batteries)
- replace the computer's cover and re-insert the screws

If you've never removed the cover of your computer before, you'll probably want to enlist the help of a PC support specialist. (Some covers are tightly fitted to the chassis of the PC, and you risk loosening some cards if you aren't gentle in removing the cover.) The entire procedure should require no more than 15 minutes.

Restoring the configuration settings

Once you've installed a new battery and replaced the computer's cover, you'll need to restore the PC's configuration settings, which were lost as soon as CMOS lost

power. To restore those settings, turn on your PC and use the same method you used before to access the configuration screen. Once that screen appears, type all the information exactly as you've recorded it. When you're finished, press the appropriate key to save your new settings and reboot. At that point, your PC should boot up and once again operate normally.

A note for IBM PS/2 users

Setting up the configuration for an IBM PS/2 computer is somewhat different than setting up most other computers. If you're using an IBM PS/2, you won't need to write down your computer's configuration settings before you change its battery. Instead, you can replace the batteries, and then let the computer's Reference Disk do the rest of the work for you. Simply insert the computer's Reference Disk into drive A and reboot the machine. When you see the message *Automatically configure the system? (Y/N)*, simply press Y. The Reference Disk is nearly always smart enough to figure out what kind of equipment is installed on the computer, and can automatically set up the proper configuration.

However, if your computer is equipped with some special peripherals and device drivers, the Reference Disk may not be able to properly configure your system automatically. Instead, you'll need to follow the instructions in your owner's manual for configuring your machine. For more information on configuring an IBM PS/2, consult your local PC support specialist.

FIND TIP

Refining files with the FIND filter

As you may know, the FIND filter will display, count, or number all lines containing (or not containing) a specified string. Recently, contributing editor David Reid sent us yet another use for the FIND filter: removing lines from a file containing (or not containing) a specified string. The technique is an excellent way of paring down a large data file to just the information you need.

For example, let's say you've exported a mailing list from a database program into an ASCII file named MAILING.TXT. Let's also suppose that the file is comma-delimited; that is, the fields of the database record are separated by commas. Each record uses one line of the file and looks like

```
Joe User,123 Main St,AnyTown,NM,12345
```

You can extract all records that contain New Mexico addresses from the mailing list by piping the file through FIND by using the command

```
C:\>type mailing.txt | find "NM,"
```

Now, let's suppose that you're interested only in records with New Mexico addresses and you want to delete all other records from the file. You can overwrite the current contents of MAILING.TXT with FIND's output by using the command

```
C:\>type mailing.txt | find "NM," > mailing.txt
```

Alternatively, you may want to remove all records with New Mexico addresses from the file. In this case, you use FIND's /V switch to exclude all lines containing the search string:

```
C:\>type mailing.txt | find /v "NM," > mailing.txt
```

Note

Some applications use both quotation marks and commas to separate fields in ASCII files. In this format, our sample record might look like the one below:

```
"Joe User", "123 Main St.", "AnyTown", "NM", "12345"
```

In this case, you can include a quotation mark in the search string by using two quotation marks in its place. For example, you can extract all records that contain New Mexico addresses from such a mailing list by issuing the command

```
C:\>type mailing.txt | find """NM"","
```

The statement is tedious to type, but logical. First, you place an additional set of quotation marks around the field shown in the record as *"NM"*. Then, you place another set of quotation marks around the *""NM""* , string. That's how you end up with the odd-looking *"""NM""*," construction.

MEM's extended memory report can be confusing

Have you ever wondered why the MEM command reports that your system has *0 bytes available contiguous extended memory*? This finding may be a surprise if you've added the line

```
device=c:\dos\himem.sys
```

to your CONFIG.SYS file. After all, you would expect HIMEM.SYS, DOS' extended memory manager, to list your system's extended memory as "available." Fortunately, the *0 bytes available* message is nothing to worry about. Basically, this line of the MEM report just allows DOS to make a fine distinction about how it handles the extended memory. To get a better idea of how MEM reports on extended memory, let's look at a sample report:

```
 655360  bytes total conventional memory
 655360  bytes available to MS-DOS
 596208  largest executable program size

3538944  bytes total contiguous extended memory
      0  bytes available contiguous extended memory
3538944  bytes available XMS memory
```

This report is from a 386 PC with 4 Mb of random access memory. The CONFIG.SYS file uses the line *device=c:\dos\himem.sys* to install the extended memory manager.

The first section of the report concerns conventional memory. As you may know, DOS considers the first 640 Kb of memory as "conventional." Since a kilobyte is 1024 bytes, MEM displays 655360—or 640 x 1024—as the *bytes total conventional memory*. The second line shows that all of this memory is available to DOS. Finally, because DOS and certain device drivers (such as ANSI.SYS) use a portion of conventional memory, we're left with 596208 bytes as the *largest executable program size*. (Loading drivers with DEVICEHIGH instead of DEVICE statements can leave you more room for the largest executable program size.)

The second section gives information about extended memory. The first line shows that we have 3538944 bytes (or 3456 Kb) of extended memory. This extended memory is the total memory of our system (4 Mb, or 4096 Kb) minus our conventional memory (640 Kb). Next comes that alarming line:

```
0  bytes available contiguous extended memory
```

On closer examination, this line also "adds up." Since HIMEM.SYS converts all available extended memory into memory meeting the XMS specification, it leaves no plain old contiguous extended memory. The last line of the MEM report confirms that this is what's happening:

```
3538944  bytes available XMS memory
```

Notes

Because we've used a bare-bones configuration in our example MEM report, your MEM report will almost certainly differ from ours. For instance, in our example, we didn't load DOS into the high memory area or create upper memory blocks. To do this, you'll need to install the EMM386.EXE driver in addition to HIMEM.SYS. You'll also need to add the line *dos=high,umb* to the CONFIG.SYS file. If you use this configuration, the first lines of your CONFIG.SYS file might look like this:

```
device=c:\dos\himem.sys
device=c:\dos\emm386.exe noems
dos=high,umb
```

(The NOEMS switch after EMM386.EXE gives you access to the upper memory blocks without creating expanded memory. Always have a boot diskette ready if you edit the CONFIG.SYS file.)

On our PC, this configuration increased the largest executable program size MEM reported to 619120—a savings of about 22 Kb over using HIMEM.SYS only. As a tradeoff, we sacrificed some of our available XMS memory. On our 4 Mb PC, the MEM command reported *3288064 bytes available XMS memory*, down from the 3538944 bytes previously available.

Of course, loading device drivers with the command DEVICEHIGH saves conventional memory, which more than makes up for the decrease in available XMS memory. For example, loading the ANSI.SYS driver with the statement

```
devicehigh=c:\dos\ansi.sys
```

saves 4 Kb of conventional memory for applications.

Thanks

Inside DOS would like to thank Craig Richmond of Princeton, New Jersey, who helped William Groce in developing the COPI.BAT batch file. We showed you how to create this batch file in the article "A Variation on Copying Miscellaneous Files to a Diskette," which appeared in last month's issue. Thanks also to Mr. Groce for pointing out our omission.

LETTERS

A frugal way to guard against running multiple copies of the DOS Shell

I read with interest the article "Preventing the DOS Shell from Using Memory It Doesn't Need," which appeared in the November 1992 issue of *Inside DOS*. Assigning the SHELL.BAT file to the DOSSHELL macro is certainly an effective means of preventing a user from accidentally running a second copy of DOSSHELL.EXE.

However, there is another method for your readers who, like me, are miserly with their hard disk storage space. If you'd like to remind yourself when you've "shelled out" to the command prompt, you might want to use this technique. It uses commands from the DOS Shell, so you won't need to create a batch file.

After you've started the DOS Shell, highlight the Command Prompt option in the Main section, either by tabbing to it or by clicking on it once. Then, select the Properties... command from the File menu by clicking it or by pressing [Alt]F,P. The DOS Shell will display the Program Item Properties dialog box. The first line displays *Command Prompt* as the program title. Now tab to or click on the second line, the Commands field. Press [Backspace] to erase the word *command*. In its place, type the following:

```
set prompt=(Shell) %prompt%; command
```

Once you've entered the new line, click OK or press [Enter]. DOS will store the new setting in the file DOSSHELL.INI.

From now on, whenever you leave the DOS Shell through the Command Prompt option, you'll see *(Shell)* in front of your regular DOS prompt. For example, if you use the common pg prompt to display the path followed by a greater-than sign, the prompt will look like this when you're in the C:\DOS directory:

```
(Shell) C:\DOS>
```

When you see this prompt, you only need to type *exit* and press [Enter] to return to the DOS Shell.

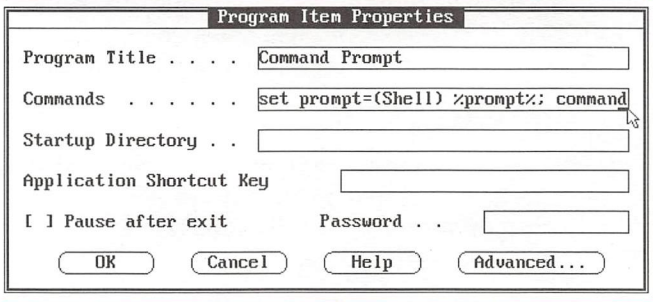
The method I've described uses no additional hard disk space because the settings for the command prompt are already stored in the DOSSHELL.INI file. Customizing the prompt will add just a few bytes to the file. On the other hand, SHELL.BAT would occupy 2048 bytes of hard disk space (one allocation unit), even though the batch file itself is only 103 bytes in size.

Norval Morgan
Moline, Illinois

Thanks for sharing your tip with us. Mr. Morgan's method offers users pressed for hard disk space an excellent way to safeguard themselves from running a second copy of DOSSHELL.COM. Placing the word *Shell* in the prompt makes it easy to remember when you need to type *exit* to return to the Shell. (Note, however, that you must type *exit*, since this technique doesn't supercede the DOSSHELL command, as the macro/batch file combination does. Should you accidentally type *dosshell* anyway—running a second copy of the program—you'll see *(Shell)(Shell)* in front of the prompt if you decide to use the Command Prompt option again.)

Changing the command prompt's properties takes just a few seconds. Figure A shows how the Program Item Properties dialog box will look after you've edited the Commands field as Mr. Morgan suggests.

Figure A



You can use Mr. Morgan's technique by editing the Program Item Properties dialog box.

Once you've made this change, the *(Shell)* reminder will appear whenever you use the Command Prompt option to temporarily exit the DOS Shell.